# U.S. PATENT APPLICATION FOR

# CACHE COHERENCY PROTOCOL WITH ORDERING POINTS


**Inventor(s):**

Stephen R. Van Doren
8 Iroquois Drive
Northborough, MA 01532

Gregory E. Tierney
161 Boston Road
Chelmsford, MA 01824

Simon C. Steely, Jr.
8 Anna Louise Drive
Hudson, NH 03051


**Attorney Docket No.: 200313588-1**

**Certificate of Mailing:**

I hereby certify that this is being deposited with the United States Postal Service as Express Mail Number EU516995825US service under 37 CFR 1.10 on January 20, 2004 and is addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Printed Name: Lisa D. Jones

Signature: 

# CACHE COHERENCY PROTOCOL WITH ORDERING POINTS

## RELATED APPLICATIONS

**[0001]** This application is related to the following commonly assigned co-pending patent applications entitled:

**[0002]** "SYSTEM AND METHOD FOR RESOLVING TRANSACTIONS IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313589-1; "SYSTEM AND METHOD TO FACILITATE ORDERING POINT MIGRATION," Attorney Docket No. 200313612-1; "SYSTEM AND METHOD TO FACILITATE ORDERING POINT MIGRATION TO MEMORY," Attorney Docket No. 200313613-1; "SYSTEM AND METHOD FOR CREATING ORDERING POINTS," Attorney Docket No. 200313614-1; "SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL WITH ORDERING POINT MIGRATION," Attorney Docket No. 200313615-1; "SYSTEM AND METHOD FOR READ MIGRATORY OPTIMIZATION IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313616-1; "SYSTEM AND METHOD FOR BLOCKING DATA RESPONSES," Attorney Docket No. 200313628-1; "SYSTEM AND METHOD FOR NON-MIGRATORY REQUESTS IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313629-1; "SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL WITH ORDERING POINT MIGRATION," Attorney Docket No. 200313630-1; "SYSTEM AND METHOD FOR CONFLICT RESPONSES IN A CACHE COHERENCY PROTOCOL," Attorney Docket No. 200313631-1; "SYSTEM AND METHOD FOR RESPONSES BETWEEN DIFFERENT CACHE COHERENCY PROTOCOLS," Attorney Docket No. 200313632-1, all of which are filed contemporaneously herewith and are incorporated herein by reference.

## BACKGROUND

**[0003]** Multi-processor systems employ two or more computer processors that can communicate with each other, such as over a bus or a general interconnect network. In such systems, each processor may have its own memory cache (or cache store) that is separate from the main system memory that the individual processors can access. Cache memory connected to each processor of the computer system can often enable fast access to data. Caches are useful because they tend to reduce latency associated with accessing data on cache hits, and they work to reduce the number of requests to system memory. In

particular, a write-back cache enables a processor to write changes to data in the cache without simultaneously updating the contents of memory. Modified data can be written back to memory at a later time.

**[0004]** Coherency protocols have been developed to ensure that whenever a processor reads a memory location, the processor receives the correct or true data. Additionally, coherency protocols help ensure that the system state remains deterministic by providing rules to enable only one processor to modify any part of the data at any one time. If proper coherency protocols are not implemented, however, inconsistent copies of data can be generated.

**[0005]** There are two main types of cache coherency protocols, namely, a directory-based coherency protocol and a broadcast-based coherency protocol. A directory-based coherency protocol associates tags with each memory line. The tags can contain state information that indicates the ownership or usage of the memory line. The state information provides a means to track how a memory line is shared. Examples of the usage information can be whether the memory line is cached exclusively in a particular processor's cache, whether the memory line is shared by a number of processors, or whether the memory line is currently cached by any processor.

**[0006]** A broadcast-based coherency protocol employs no tags. Instead, in a broadcast-based coherency protocol, each of the caches monitors (or snoops) requests to the system. The other caches respond by indicating whether a copy of the requested data is stored in the respective caches. Thus, correct ownership and usage of the data are determined by the collective responses to the snoops.

## SUMMARY

**[0007]** One embodiment of the present invention may comprise a system that includes a first node having an associated cache including data having an associated first cache state. The first cache state is capable of identifying the first node as being an ordering point for serializing requests from other nodes for the data.

**[0008]** Another embodiment of the present invention may comprise a multi-processor network that includes a plurality of processor nodes. Each processor node has at least one associated cache. The plurality of processor nodes employs a coherency protocol. The coherency protocol employs ordering points for serializing requests for data associated with the at least one associated cache of the plurality of processor nodes. The

ordering point for the data is capable of being associated with the at least one associated cache of one of the processor nodes.

**[0009]** Another embodiment of the present invention may comprise a method that includes employing a first cache state to identify a first node of a system as being an ordering point for a block of data. The method also includes providing a data response from the first node to a request from a second node of the system for the block of data.

**[0010]** Another embodiment of the present invention may comprise a coherency protocol operative to assign a cache state to a cache line of one node of a plurality of nodes of a system. The cache state defines the one node as an ordering point in the system for data in the cache line of the one node.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0011]** FIG. 1 depicts an example of a multi-processor system.

**[0012]** FIG. 2 depicts an example of another multi-processor system.

**[0013]** FIG. 3 depicts an example of a processor within a multi-processor system.

**[0014]** FIG. 4 depicts a timing diagram for a first example scenario illustrating state transitions for a coherency protocol.

**[0015]** FIG. 5 depicts a timing diagram for a second example scenario illustrating state transitions for a coherency protocol.

**[0016]** FIG. 6 depicts a timing diagram for a third example scenario illustrating state transitions for a coherency protocol.

**[0017]** FIG. 7 depicts a timing diagram for a fourth example scenario illustrating state transitions for a coherency protocol.

**[0018]** FIG. 8 depicts a timing diagram for a fifth example scenario illustrating state transitions for a coherency protocol.

**[0019]** FIG. 9 depicts a flow diagram illustrating a method.

## DETAILED DESCRIPTION

**[0020]** This disclosure relates generally to systems and methods for a cache coherency protocol in a system including a plurality of processor nodes. The cache coherency protocol supports cache ordering points for data blocks cached at the processor nodes of the system. A state of the data block cached at a processor node defines the processor node as a point for serializing requests from other processor nodes for the data block. Thus, when a source processor node issues a request for a data block, a target

processor node that is a cache ordering point for the data can provide a data response. This can be referred to as a "two-hop" transaction, the first hop being the request issued to the target processor and the second hop being the target processor providing the data response to the source processor. This can provide improved latency over systems that require three hops to complete a transaction.

**[0021]** The cache coherency protocol also supports migratory data. Migratory data can be defined as a class of data blocks that are shared by a plurality of processors and are characterized by a per processor reference pattern which includes a read and a write in close temporal proximity in the instruction stream of the processor. In general, migratory data blocks are expected to be shared in a manner in which each processor is provided with an opportunity to complete its read-write sequence before any other processors initiate their read-write sequence. The cache coherency protocol supports migratory data *via* the transfer of cache ordering points for the data between the processors.

**[0022]** Migratory data support is a system function implemented through the cache coherency protocol described herein. When a processor begins a migratory data sequence, for example, it encounters a read reference in its instruction stream that provides no indication that the read reference is operative on a migratory data line. Thus, when a read request is issued to the system by the processor, it is issued as a simple read request (XREAD). The cache coherency protocol described herein employs a predictive function to determine whether the read request is addressing a migratory data line. This predictive function can be a cache state decode mechanism responsive to the states of all processors that are targets of snoops associated with the XREAD request. The predictive function implemented in the cache coherency protocol described herein covers a comprehensive set of coherence timing and conflict cases that can arise during migratory data flows. Once migratory data is detected, the XREAD request is completed implementing measures to ensure that the XREAD is completed correctly.

**[0023]** If a read snoop finds the requested line cached in a modified state in another cache, the data is returned to the requestor in a dirty state. Thus, in effect, when migration takes place, the migratory read command acts like a write command, moving the cache ordering point to the requesting node. The cache coherency protocol also employs conflict states that are assigned to a miss address file (MAF) entry for an outstanding broadcast snoop request. The conflict states are used to determine how to handle conflicts between requests for the same cache line at essentially the same time that are initiated by more than one processor and recorded in MAF entries at those processors.

**[0024]** FIG. 1 depicts an example of a system 10 implementing a cache coherency protocol. The system 10 illustrates a multi-processor environment that includes a plurality of processors 12 and 14 (indicated at PROCESSOR 1 through PROCESSOR N, where N is a positive integer (N>1)). The system 10 also includes memory 16 that provides a shared address space. For example, the memory 16 can include one or more memory storage devices (*e.g.*, dynamic random access memory (DRAM)).

**[0025]** The processors 12 and 14 and memory 16 define nodes in the system 10 that can communicate with each other *via* requests and corresponding responses through a system interconnect 18. For example, the system interconnect 18 can be implemented as a switch fabric or a hierarchical switch. Also associated with the system 10 can be one or more other nodes, indicated schematically at 20. The other nodes 20 can correspond to one or more other multi-processor systems connected to the system interconnect 18, such as through an appropriate interconnect interface (not shown).

**[0026]** Each of the processors 12 and 14 includes at least one corresponding cache 22 and 24. For purposes of brevity, each of the respective caches 22 and 24 is depicted as a unitary memory device, although the caches may include a plurality of memory devices or different cache levels. The system 10 employs the caches 22 and 24 and the memory 16 to store blocks of data, referred to herein as "memory blocks." A memory block can occupy part of a memory line, an entire memory line or span across multiple lines. For purposes of simplicity of explanation, however, it will be assumed that a memory block occupies a single "memory line" in memory or a "cache line" in a cache. Additionally, a given memory block can be stored in a cache line of one or more caches as well as in a memory line of the memory 16.

**[0027]** Each of the caches 22 and 24 includes a plurality of cache lines. Each cache line has an associated tag address that identifies corresponding data stored in the line. Each cache line can also include information identifying the state of the data stored in the respective cache. A given memory block can be stored in a cache line of one or more of the caches 22 and 24 as well as in a memory line of the memory 16, depending on the state of the line. Whether a cache line contains a coherent copy of the data also depends on the state of the cache line. Certain states employed by the coherency protocol can define a given cache line as an ordering point for the system 10 employing a broadcast-based protocol. An ordering point characterizes a serialization of requests to the same memory line (or memory block) that is understood and followed by the system 10.

**[0028]** The system 10 implements the cache coherency protocol to manage the sharing of memory blocks to help ensure coherence of data. The coherency protocol establishes rules for transitioning between states, such as if data is read from or written to memory 16 or one of the caches 22 and 24.

**[0029]** The coherency protocol can be implemented as a hybrid cache coherency protocol, such as a broadcast source snoop protocol (SSP) implemented in conjunction with a forward progress (*e.g.*, directory-based or null-directory) protocol (FPP). A forward progress protocol can be considered deterministic because FPP transactions (*e.g.*, FPP request transactions) are typically resolved successfully. A source snoop protocol can be considered non-deterministic because, while most SSP transactions (*e.g.*, SSP broadcast snoop request transactions) are resolved successfully, situations may be encountered where an SSP transaction cannot be deterministically resolved. Thus, the SSP protocol may be considered an incomplete protocol since there are circumstances that require the use of a correctness protocol (*e.g.* a forward progress protocol) in which all transactions can be deterministically resolved. Characteristic of the hybrid cache coherency protocol, transactions, such as requests for data, are initially transmitted using an SSP broadcast snoop request. If the SSP broadcast snoop request cannot resolve the transaction deterministically, such as where there is a conflict between multiple processors attempting to read and/or write the same cache line, the hybrid cache coherency protocol can transition to an FPP mode to deterministically resolve the transaction. Other forward progress techniques could also be utilized.

**[0030]** As used herein, a node that issues a request, such as a read or write request, defines a source node. Other nodes within the system 10 are potential targets of the request. Additionally, each memory block in the system 10 can be assigned a home node that maintains necessary global information and a data value for that memory block. When a source node issues a source broadcast snoop request for data, an entry associated with the request is allocated in a miss address file (MAF). The MAF maintains information associated with, for example, the address of the data being requested, the type of request, and response information received from other nodes in response to the request. The MAF entry for the request is maintained until the request associated with the MAF is complete.

**[0031]** For example, when a source node, such as the processor 12, requires a copy of a given memory block, the source node typically first requests the memory block from its local, private cache by comparing selected cache tags to the address associated with the

memory block. If the data is found locally, the memory access is resolved without communication *via* the system interconnect 18. When the requested memory block is not found locally, the source node 12 can request the memory block from the system 10, including the memory 16. In addition to the request identifying an address associated with the requested memory block, the request usually identifies the type of request or command being issued by the requester. Whether the other nodes 14 and the memory 16 will return a response depends upon the type of request, as well as the state of the identified memory block if contained in the responding nodes. The cache coherency protocol implemented by the system 10 defines the available states and possible state transitions.

**[0032]** A set of cache states that can be included in the cache coherency protocol described herein is depicted below in Table 1. Each cache line of the respective caches 22 and 24 of the processors 12 and 14 may be associated or tagged with one of the cache states in table 1. Since there are eight possible states, the state information can be encoded by a three-bit data word, for example.

**TABLE 1**

| STATE | DESCRIPTION |
|---|---|
| I | Invalid - The cache line does not exist. |
| S | Shared - The cache line is valid and unmodified by caching processor. Other processors may have valid copies, and the caching processor cannot respond to snoops by returning data. |
| E | Exclusive - The cache line is valid and unmodified by caching processor. The caching processor has the only cached copy in the system and may respond to snoops by returning data. |
| F | First (among equals) - The cache line is valid and unmodified by caching processor. Other processors may have valid copies, and caching processor may respond to snoops by returning data. |
| D | Dirty - The cache line is valid and more up-to-date than memory. The cache line has not been modified by the caching processor, and the caching processor has the only cached copy in the system. The caching processor responds to snoops by returning data and writes data back to memory upon displacement. The dirty state permits a modified block to be transferred between caches without updating memory. |
| M | Modified - The cache line is valid and has been modified by the caching processor. The caching processor has the only cached copy in the system, and the caching processor responds to snoops by returning data and writes data back to memory upon displacement. |
| O | Owned - The cache line is valid and more up-to-date than memory. The caching processor cannot modify the cache line. Other processors may have valid copies, and the caching processor responds to snoops by returning data and writes data back to memory upon displacement. |
| T | Transition - The cache line is in transition. The cache line may be transitioning from O, M, E, F or D to I, or the line may be transitioning from I to any one of the valid states. |

**[0033]** As mentioned above, the state of a cache line can be utilized to define a cache ordering point in the system 10. In particular, for a protocol implementing the states set forth in Table 1, a processor including a cache line having one of the states M, O, E, F or D may be referred to as an owner processor or node and can serve as a cache ordering point for the data contained in the cache line for transactions in the broadcast-based protocol. An owner processor (*e.g.*, processor 12 or 14) that serves as the cache ordering point is capable of responding with data to snoops for the data. For example, processor 14 may be an owner processor for particular data and thus can provide a copy of the data to another cache 12 in response to a read request from cache 12 in a two-hop transaction. This can improve latency of the transaction over that which would be experienced in a

three-hop transaction, *e.g.*, where the request is sourced to a home processor, which directs the request to an owner processor to obtain the data. The type of data returned by an owner processor depends on the state of the data stored in the processor's cache. The response may also vary based on the type of request as well as whether a conflict exists. The memory 16 seeks to return a copy of the data stored in the memory. The memory copy of the data is not always a coherent copy and may be stale (*e.g.*, when there is a modified copy of the data cached by another processor).

**[0034]** The cache coherency protocol described herein also provides for ordering point migration in which a cache ordering point is transferred from cache of a target processor to cache of a source processor in response to a source broadcast read request depending on a migratory predictor, such as the cache state of a target processor for a line of data. For example, a target node (*e.g.*, processor 14) including an M-state cache line can, in response to a source broadcast read request, provide an ownership data response to a source node (*e.g.*, processor 12), and the source node cache line transitions to the D-state. Upon completion of the ordering point transfer, the target processor 14 cache line transitions to the I-state. The ordering point is thus transferred (*i.e.*, the ordering point migrates) from the target processor 14 to the source processor 12.

**[0035]** To mitigate the vulnerability of the ordering point during migration, the cache line of the target processor 14 can transition to the T-state while the ordering point migration is pending. Additionally, the source processor 12 can provide a message that acknowledges when the ordering point has successfully migrated (*e.g.*, a migration acknowledgement or "MACK" message). The cache line of the target processor 14 can further transition from the T-state to the I-state in response to receiving the MACK message from the source processor 12. The target processor 14 can respond to the MACK message by providing a further acknowledgement message back to the source processor 12 (*e.g.*, a MACK acknowledgement or MACK-ACK message). The source broadcast read request by the source processor 12 that initiated the migration sequence can be considered completed in response to receiving the MACK-ACK message from the target processor 14.

**[0036]** The processors 12 and 14 of the system 10 can obtain copies of desired data by issuing data requests in either the SSP or FPP portion of the cache coherency protocol implemented in the system. A list of example data requests that can be included in the SSP portion of the cache coherency protocol described herein, and thus issued through a

source broadcast request by a processor (*e.g.*, processors 12 and 14), is depicted below in Table 2.

**TABLE 2**

| Request Type | Request | Request Description |
|---|---|---|
| Reads | XREADN | Broadcast read line code: Non-migratory read request. |
| | XREAD | Broadcast read line data: Simple read request. |
| | XREADC | Broadcast read current (non-coherent read). |
| Writes | XRDINVAL | Broadcast read and invalidate line with owner. |
| | XUPGRADE | Broadcast invalidate line - upgrade un-writable copy. |
| Memory Updates | XMWRITE | Broadcast memory write-back - victim write. |
| | XMUPDATE | Broadcast memory update - victim write. |
| | XWRITEC | Broadcast write coherent. |
| Special Commands | MACK | Broadcast migration acknowledgment. |
| | XINVAL | Broadcast invalidate. |

**[0037]** According to the cache coherency protocol described herein, source processors 12 and 14 issue data requests initially as broadcast snoop requests using the SSP commands set forth in Table 2. If the snoop requests fail and a transition to the FPP is required (*e.g.*, due to a conflict), the system 10 can reissue the request using a forward progress technique. For example, the system 10 can transition to an FPP mode and the requests can be reissued using FPP commands.

**[0038]** Whenever a broadcast read or write snoop is issued by a source node (*e.g.*, source processor 12) in the system 10, target nodes of the system (*e.g.*, target processor 14, memory 16, and nodes 20) may issue an SSP response to the snoop. A list of example SSP responses that may be included in the cache coherency protocol described herein is depicted below in Table 3.

**TABLE 3**

| SSP Broadcast Response | Response Description |
|---|---|
| D-DATA | Ownership data response - Corresponding snoop command was the first to arrive at a cache ordering point (M, O, D, E, F state); the ordering point is being transferred to the requesting processor. At most, one D-DATA command can exist per cache line at any given time. |
| S-DATA | Shared data response - Data is being returned from a cache ordering point; the ordering point is not being transferred. |
| M-DATA | Memory data response - Data is being returned from home memory. |
| MISS | General snoop response:<br>- Snoop failed to match a cache or MAF entry at a snoop target.<br>- Snoop matched at a snoop target and invalidated a cache line at the target.<br>- Acknowledgement for broadcast invalidate line requests.<br>- Acknowledgement for broadcast migration acknowledgement requests.<br>- Acknowledgement for broadcast victim write requests. |
| SHARED | Snoop hit shared - Read snoop matched on a cache line in the S-state. |
| CONFLICT | Snoop conflict - Snoop matched a valid MAF (read or write) or T-state cache line at a target processor. |
| RD-CONF | Snoop read conflict - A special case conflict used where a snoop matched a valid read MAF. |
| FPP | Snoop hit FPP-Mode MAF - Some other processor is trying to access the same cache line and has already transitioned to the forward progress protocol (FPP) mode. This response is required for forward progress/starvation avoidance. |

**[0039]** When a source node (*e.g.*, source processor 12) issues a source broadcast request for data, each of the target nodes (*e.g.*, target processor 14, memory 16, and nodes 20) having a copy of the requested data may provide a data response. In the cache coherency protocol described herein, there are three different types of data responses: shared data responses (S-DATA), dirty data responses (D-DATA), and memory data responses (M-DATA). It is thus possible that, in response to a source broadcast request for data, the source processor 12 can receive several different data responses.

**[0040]** In general, D-DATA overrides both M-DATA and S-DATA, meaning that D-DATA will result in a cache fill, overwriting M-DATA or S-DATA. Additionally, S-DATA will overwrite M-DATA, but not D-DATA. Thus, D-DATA has priority over M-

DATA and S-DATA, and S-DATA has priority over M-DATA. M-DATA results in a cache fill only if no other types of data have been received.

**[0041]** Examples of processor snoop responses to source broadcast snoop requests that can occur in the system 10 and the target node state transitions that result therefrom are provided in Table 4. The state transitions set forth in Table 4 assume that no conflicts are encountered in response to the respective commands. Conflict conditions can affect state transitions. As shown in Table 4, the response to the source node varies depending on the type of broadcast snoop request received at the target node and the cache state of the target node when the snoop request is received.

**TABLE 4**

| Source Node Request | Target Node Cache State | Target Node Next Cache State | Response to Source Node |
|---|---|---|---|
| XREAD | T | Unchanged | CONFLICT |
| XREAD | I | Unchanged | MISS |
| XREAD | S | Unchanged | Shared |
| XREAD | E, F | F | S-DATA |
| XREAD | D, O | O | S-DATA |
| XREAD | M | T | D-DATA |
| XRDINVAL | T | Unchanged | CONFLICT |
| XRDINVAL | S, I | I | MISS |
| XRDINVAL | M, D, O, E, F | T | D-DATA |

**[0042]** A target node can provide an ownership data response that includes D-DATA, for example, when the processor has an ownership state (*e.g.*, M, O, E, F or D) associated with the cached data in the SSP protocol. It is the state of the cached data that defines the node (processor) as a cache ordering point for the data. When a processor responds with D-DATA, the ordering point is transferred to the requesting processor. S-DATA is a shared data response that indicates data is being returned from a cache ordering point, although the ordering point itself is not being transferred to the requester. An S-DATA response also indicates that a copy of the data may be in one or more other caches. An M-DATA response can be provided by memory (*e.g.*, a home node) by returning the present value for the data stored in memory. It is possible that the M-DATA is stale and not up-to-date. Examples of state transitions that can occur during selected source broadcast transactions in the system 10 for selected processor commands are provided in Table 5.

**TABLE 5**

| Command | Current State | | | Next State | | |
|---|---|---|---|---|---|---|
| | Source | Owner | Sharer | Source | Owner | Sharer |
| Broadcast migratory read request XREAD | I | I | I | E | I | I |
| | I | I | S | F | I | S |
| | I | E | I | S | F | I |
| | I | F | I | S | F | I |
| | I | F | S | S | F | S |
| | I | D | I | S | O | I |
| | I | M | I | D | I | I |
| | I | O | I | S | O | I |
| | I | O | S | S | O | S |
| Broadcast read and invalidate line with owner XRDINVAL | I | I | I | E | I | I |
| | I | I | S | E | I | I |
| | I | E | I | E | I | I |
| | I | F | I | E | I | I |
| | I | F | S | E | I | I |
| | I | D | I | D | I | I |
| | I | M | I | D | I | I |
| | I | O | I | D | I | I |
| | I | O | S | D | I | I |

**[0043]** By way of further example, with reference to FIG. 1, assume that the processor 12 (a source node) requires a copy of data associated with a particular memory address, and assume that the data is unavailable from its own local cache 22. Since the processor 12 does not contain a copy of the requested data, the cache line of the processor may be initially in the I-state (invalid) for that data or it may contain different data altogether. For purposes of simplicity of explanation, the starting state of the source node cache line for this and other examples is the I-state. The processor 12, operating as the source node, transmits a source broadcast read snoop (XREAD) to the system 10, including the other processor 14, to the memory 16, and to the other nodes 20 *via* the system interconnect 18.

**[0044]** In this example, it is assumed that, at the time of the XREAD request, processor 14 is an owner node, *i.e.*, a cache ordering point for the data. Assume also that the other nodes 20 are invalid for the data. For this example, assume that the processor 14 has a copy of the data in an M-state cache line of cache 24. Referring to Table 5, after the XREAD request transaction broadcast from the source processor 12 is complete, the source processor 12 will be in the D-state for the data, having received a D-DATA response from the owner processor 14 (see, *e.g.*, Table 4). The processor 14 will be in the I-state for the data when the migratory data transaction is completed. The processor 12,

serving as the cache ordering point for the data, thus becomes a point for serializing subsequent requests for the data and may transfer the ordering point to subsequent requesting processors without updating memory 16.

**[0045]** FIG. 2 depicts an example of a multi-processor computing system 50. The system 50, for example, includes an SMP (symmetric multi-processor) processor 52 that includes processors (P1, P2, P3, P4) 54, 56, 58 and 60 in communication with each other *via* an interconnect 62. The interconnect 62 facilitates transferring data between processors and memory of the system 50. While four processors 54, 56, 58, and 60 are depicted in the example of FIG. 2, those skilled in the art will appreciate that a greater or smaller number of processors can be implemented in the processor 52.

**[0046]** Each processor 54, 56, 58, and 60 also includes an associated cache 64, 66, 68 and 70. The caches 64, 66, 68, and 70 can enable faster access to data than from an associated main memory 72 of the processor 52. The system 50 implements a cache coherency protocol designed to guarantee coherency of data in the system. By way of example, the cache coherency protocol can be implemented to include a source broadcast protocol in which broadcast snoops or requests for data are transmitted directly from a source processor to all other processors and memory in the system 50. The source broadcast protocol can further be implemented in conjunction with another forward progress protocol, such as a null-directory or other directory-based protocol. The system 50 of FIG. 2, for example, employs the source broadcast protocol to process a request for data. If the request cannot be processed using the source broadcast protocol, such as where a conflict exists, the system 50 transfers to its forward progress protocol.

**[0047]** The memory 72 can include multiple memory modules (M1, M2, M3, M4) 74, 76, 78 and 80. For example, the memory 72 can be organized as a single address space that is shared by the processors 54, 56, 58 and 60 as well as other nodes 82 of the system 50. Each of the memory modules 74, 76, 78 and 80 can include a corresponding directory 84, 86, 88 and 90 that defines where the corresponding coherent copy of the data should reside in the system 50. Alternatively, the memory modules may contain no directories. A coherent copy of data, for example, may reside in a home node (*e.g.*, associated with a given memory module) or, alternatively, in a cache of one of the processors 54, 56, 58 and 60.

**[0048]** The other node(s) 82 can include one or more other SMP nodes associated with the SMP processor 52 *via* the interconnect 62. For example, the interconnect 62 can be implemented as a switch fabric or hierarchical switch programmed and/or configured to

manage transferring requests and responses between the processors 54, 56, 58, and 60 and the memory 70, as well as those to and from the other nodes 82.

**[0049]** When the processor 54 requires desired data, the processor 54 operates as a source and issues a source broadcast snoop (*e.g.*, a broadcast read or broadcast write request) to the system 50, including all other processors 56, 58 and 60 as well as to memory 72, *via* the interconnect 62. The cache coherency protocol described herein is designed to ensure that a correct copy of the data is returned in response to the source broadcast snoop.

**[0050]** By way of further example, with reference to FIG. 2, assume that the processor 54 (a source processor) requires a copy of data associated with a particular memory address, and assume that the data is unavailable from its own local cache 64. Since the source processor 54 does not contain a copy of the requested data, the cache line of the source processor may be initially in the I-state (invalid) for that data or it may contain different data altogether. The source processor 54 transmits a source broadcast read snoop (XREAD) to the system 50, including the processors 56, 58, and 60, as well as to the memory 72 and the other nodes 82, *via* the system interconnect 62.

**[0051]** In this example, it is assumed that, at the time of the XREAD request, processor 56 is an owner processor, *i.e.*, a cache ordering point for the data. Assume also that processors 58 and 60, and the other nodes 82 are invalid for the data. For this example, assume that the owner processor 56 has a copy of the data in an F-state cache line of cache 66. Referring to Table 5, after the XREAD request transaction broadcast from the source processor 54 is complete, the source processor 54 will be in the S-state for the data, having received an S-DATA response from the owner processor 56 (see, *e.g.*, Table 4). In this example, the cache ordering point for the data remains at the owner processor 56. The source processor 54 receives a shared copy of the data without updating memory 72.

**[0052]** FIG. 3 depicts an example of another multi-processor system 100 that includes a plurality of processors 102, 104 and 106 in communication with each other *via* a switch fabric 108. The system 100 also includes associated memory 110, which can be organized as a single address space that is shared by the processors 102, 104, and 106. For example, the memory 110 can be implemented as a plurality of separate memory modules associated with each of the respective processors 102, 104, and 106 for storing data. The system 100, for example, can be implemented as an integrated circuit or as circuitry containing plural integrated circuits.

**[0053]** The system 100 can employ a source broadcast or source-snoopy cache coherency protocol. For this type of protocol, a source processor 102, 104, and 106 can issue a source broadcast request to all other processors in the system and to the memory 110. In the event that conflict arises, or the source broadcast request otherwise fails, the system 100 can transfer to a forward-progress protocol, such as a null-directory or other directory-based protocol.

**[0054]** In a null-directory-based protocol, for example, the memory 110 includes home nodes for each cache line. Instead of issuing a broadcast to all cache targets, the source issues a single request to the home node for such data. The home node thus operates as static ordering point for requested data since all requests are sent to the home node for ordering before snoops are broadcast. This tends to add an additional hop for the majority of references compared with a broadcast-based protocol described above. If the system employs a standard directory-based protocol, ordering is implemented, but the memory 110 employs associated directories that facilitate locating the data (*e.g.*, based on the directory state associated with the requested data). In a standard directory protocol, there will be times when the directory can indicate that there are no cached copies, and thus the home node can respond with the data without issuing any snoops to the system 100.

**[0055]** The processor 102 includes cache memory 114 that contains a plurality of cache lines 116 (*e.g.*, lines 1-M, where M is a positive integer, $M \geq 1$). Each cache line 116 can contain one or more memory blocks. A tag address (ADDRESS) is associated with the data contained in each cache line 116. Additionally, each cache line 116 can contain state information identifying the state of the data contained at that cache line. Examples of states that can be associated with each cache line 116 are identified above in Table 1.

**[0056]** A cache controller 118 is associated with the cache memory 114. The cache controller 118 controls and manages access to the cache memory, including requests for data and responses. The cache controller 118 communicates requests and responses *via* a switch interface 120 that is coupled with the switch fabric 108. The switch interface 120, for example, includes an arrangement of queues (*e.g.*, input and output queues) or other data structures that organize both requests and responses issued by the processor 102 as well as requests and responses for execution by the processor.

**[0057]** In the example of FIG. 3, the cache controller 118 includes a state engine 122 that controls the state of each respective line 116 in the cache memory 114. The state engine 122 is programmed and/or configured to implement state transitions for the cache lines 116 based on predefined rules established by the cache coherency protocol described herein. For example, the state engine 122 can modify the state of a given cache line 116 based on requests issued by the processor 102. Additionally, the state engine 122 can modify the state of a given cache line 116 based on responses received at the processor 102 for the given tag address, such as may be provided by another processor 104, 106 and/or the memory 110.

**[0058]** The cache controller 118 also includes a request engine 124 that sends requests to the system 100. The request engine 124 employs a miss address file (MAF) 126 that contains MAF entries for outstanding requests associated with some subset of the locations in the cache memory 114. The MAF can be implemented as a table, an array, a linked list or other data structure programmed to manage and track requests for each cache line. For example, when the processor 102 requires data associated with a given address for a given line 116, the request engine 124 creates a corresponding entry in the MAF 126. The MAF entry includes fields that identify, for example, the address of the data being requested, the type of request, and response information received from other nodes in response to the request. The request engine 124 thus employs the MAF 126 to manage requests issued by the processor 102 as well as responses to such requests. The request engine can employ a data state machine and conflict state machine (see, *e.g.*, FIGS. 2 and 3) associated with each MAF entry for helping to manage a data state and a conflict state associated with each MAF entry.

**[0059]** The cache controller 118 also includes a response engine 128 that controls responses provided by the processor 102. The processor 102 provides responses to requests or snoops received *via* the switch interface 120 from another processor 104 and 106 or memory 110. The response engine 128, upon receiving a request from the system 100, cooperates with the state engine 122 and the MAF 126 to provide a corresponding response based on the type of request and the state of data contained in the cache memory 114. For example, if a MAF entry exists for a tag address identified in a request received from another processor 104, 106 or memory 110, the cache controller 118 can implement appropriate conflict resolution defined by the coherency protocol. The response engine 128 thus enables the cache controller to send an appropriate response to requesters in the

system 100. A response to a request can also cause the state engine 122 to effect a state transition for an associated cache line 116.

**[0060]** By way of example, assume that the processor 102 requires data not contained locally in its cache memory 114. The request engine 124 will create a MAF entry in the MAF 126, corresponding to the type of request and the tag address associated with data required. For a read request, for example, the processor 102 issues an XREAD request and allocates a corresponding entry in the MAF 126. In this example, it is assumed that, at the time of the XREAD request, processor 104 is an owner processor, *i.e.*, a cache ordering point for the data. Assume also that processor 106 is invalid for the data. For this example, assume that the owner processor 104 has a copy of the data in a D-state cache line. Referring to Table 5, after the XREAD request transaction broadcast from the source processor 104 is complete, the source processor 104 will be in the S-state for the data, having received an S-DATA response from the owner processor 106 (see, *e.g.*, Table 4). In this example, processor 106 transitions to the O-state (see, *e.g.*, Table 5) and, thus, the cache ordering point for the data remains at the processor 106. The source processor 104 receives a shared copy of the data without updating memory 110.

**[0061]** Those skilled in the art will understand and appreciate that the protocol described herein can be implemented in multi-processor systems employing general system interconnects (*e.g.*, as contrasted with a bus-based interconnect) where all processors and memory are on the same bus. In prior bus-based systems, all observers of broadcasts from different sources in the system perceive the broadcasts in the same order that the broadcasts occurred on the bus, as the bus itself operates as an ordering point. In contrast, the protocol described herein can be implemented over a general system interconnect (*e.g.*, an unordered network interconnect) in which each observer of the broadcasts from plural different sources can perceive the order of the broadcasts differently.

**[0062]** In view of the foregoing structural and functional features described above, certain methods that can be implemented using a coherency protocol will be better appreciated with reference to FIGS. 4-9. FIGS. 4-8 depict various example timing diagrams for transaction scenarios that can arise in a multi-processor system employing a cache coherency protocol as described herein. Each of the examples illustrates various interrelationships between requests and responses and state transitions that can occur for a given memory tag address in different memory devices or caches. In each of these examples, time flows in the direction of an arrow labeled "TIME." Those skilled in the art

may appreciate that there can be various other scenarios or transactions that can be implemented in a multi-processor system employing a cache coherency protocol as described herein.

**[0063]** FIG. 4 illustrates a network 160 that includes processor nodes 162, 164, and 166 and a home node 168. Initially, nodes 162, 164, and 166 are in an I-state for a particular cache line. Node 168 contains a memory copy of the data associated with the cache line. In this example case, node 162 allocates a RDMAF entry 170 for the requested data and broadcasts an XREAD request to nodes 164 and 166, which, being in the I-state, each return a MISS response (see, *e.g.*, Table 4). Next, node 162 receives an M-DATA response to an XREAD request broadcast from node 162 to home node 168. Node 162 transitions to the E-state upon receiving the M-DATA response from node 168 because, at this point, MISS responses have been received from all of the nodes to which node 162 broadcast the XREAD snoop request. Thus, in the example of FIG. 4, node 162 receives the data from memory because there are no other cache ordering points from which to receive the data. Node 162, being in the E-state for the data, has an exclusive copy of the data and is a cache ordering point for the data. Node 162 may respond to subsequent requests for the data in accordance with the cache coherency protocol as described herein.

**[0064]** FIG. 5 illustrates a network 180 that includes processor nodes 182, 184, and 186 and a home node 188. Initially, nodes 182 and 184 are in an I-state for a particular cache line. Home node 186 is in the O-state for the cache line and thus is a cache ordering point for the data. Node 188 contains a memory copy of the data associated with the cache line. In this example case, node 182 allocates a RDMAF entry 190 for the requested data and broadcasts an XREAD request to nodes 184, which, being in the I-state, each return a MISS response (see, *e.g.*, Table 4). Next, node 182 receives an S-DATA response to an XREAD request broadcast from node 182 to node 186 because node 186 is a cache ordering point for the data. Node 186 remains in the O-state for the data. Next, node 182 receives an M-DATA response to an XREAD request broadcast from node 182 to home node 188.

**[0065]** At this point, responses have been received from all of the nodes to which node 182 broadcast the XREAD snoop request. Node 182, having received M-DATA and S-DATA, fills the cache at node 182 with the S-DATA because, according to the cache coherency protocol, S-DATA overrides M-DATA. Node 182 transitions to the S-state and becomes a sharer node for the data. Thus, in the example of FIG. 5, node 182 receives the data from node 186 because node 186 is a cache ordering point for the data. Node 186

remains the cache ordering point for the data and thus may respond to subsequent requests for the data in accordance with the cache coherency protocol as described herein.

**[0066]** FIG. 6 illustrates a network 200 that includes processor nodes 202, 204, and 206 and a home node 208. Initially, nodes 202 and 204 are in an I-state for a particular cache line. Node 206 is in the M-state for the cache line and thus is a cache ordering point for the data. Home node 208 contains a memory copy of the data associated with the cache line. In this example case, node 202 allocates a RDMAF entry 210 for the requested data and broadcasts an XREAD request to node 204, which, being in the I-state, returns a MISS response. Next, node 202 receives a D-DATA response to an XREAD request broadcast from node 202 to node 206 because node 206 is a cache ordering point for the data (see, *e.g.*, Table 4). Node 206 transitions to the T-state for the data. Next, node 202 receives an M-DATA response to an XREAD request broadcast from node 202 to home node 208.

**[0067]** At this point, responses have been received from all of the nodes to which node 202 broadcast the XREAD snoop request. Node 202, having received M-DATA and D-DATA, fills the cache at node 202 with the D-DATA because, according to the cache coherency protocol, D-DATA overrides M-DATA. In this instance, the ordering point for the modified data at node 206 migrates to node 202 without updating memory at home node 208. Node 202 transitions to the D-state and initiates a MACK/MACK-ACK sequence with node 206 to complete the migratory read transaction. Thus, in the example of FIG. 6, node 202 receives the migratory data from node 206 and becomes the cache ordering point for the data. Node 202 thus may respond to subsequent requests for the data in accordance with the cache coherency protocol as described herein.

**[0068]** FIG. 7 illustrates a network 220 that includes processor nodes 222, 224, and 226 and a home node 228. Initially, nodes 222 and 224 are in an I-state for a particular cache line. Node 226 is in one of the E and F states for the cache line and thus is a cache ordering point for the data. Node 228 contains a memory copy of the data associated with the cache line. In this example case, node 222 allocates a write MAF (WRMAF) entry 230 for the requested data and broadcasts an XRDINVAL request to node 224, which, being in the I-state, returns a MISS response (see, *e.g.*, Table 4). Next, node 222 receives an S-DATA response to an XRDINVAL request broadcast from node 222 to node 226 because node 226 is a cache ordering point for the data. Node 226 transitions to the I-state for the data due to the invalidate line functionality of the XRDINVAL command. Next,

node 222 receives an M-DATA response to an XRDINVAL request broadcast from node 222 to home node 228.

**[0069]** At this point, responses have been received from all of the nodes to which node 222 broadcast the XRDINVAL snoop request. Node 222, having received M-DATA and S-DATA, fills the cache at node 222 with the S-DATA because, according to the cache coherency protocol, S-DATA overrides M-DATA. Node 222 transitions to the E-state for the data and thus has an exclusive copy of the data. Thus, in the example of FIG. 7, node 222 becomes the cache ordering point for the data. Node 222 thus may respond to subsequent requests for the data in accordance with the cache coherency protocol as described herein.

**[0070]** FIG. 8 illustrates a network 240 that includes processor nodes 242, 244, and 246 and a home node 248. Initially, nodes 242 and 244 are in an I-state for a particular cache line. Node 246 is in one of the M, O, and D states for the cache line and thus is a cache ordering point for the data. Node 248 contains a memory copy of the data associated with the cache line. In this example case, node 242 allocates a WRMAF entry 250 for the requested data and broadcasts an XRDINVAL request to node 244, which, being in the I-state, returns a MISS response. Next, node 242 receives a D-DATA response to an XRDINVAL request broadcast from node 242 to node 246 because node 246 is a cache ordering point for the data (see, *e.g.*, Table 4). Node 246 transitions to the T-state for the data. Next, node 242 receives an M-DATA response to an XRDINVAL request broadcast from node 242 to home node 248.

**[0071]** At this point, responses have been received from all of the nodes to which node 242 broadcast the XRDINVAL snoop request. Node 242, having received M-DATA and D-DATA, fills the cache at node 242 with the D-DATA because, according to the cache coherency protocol, D-DATA overrides M-DATA. In this instance, the ordering point for the modified data at node 246 migrates to node 242 without updating memory at home node 248. Node 242 transitions to the D-state and initiates a MACK/MACK-ACK sequence with node 246 to complete the migration transaction. Thus, in the example of FIG. 8, node 242 receives the migratory data from node 246 and becomes the cache ordering point for the data. Node 242 thus may respond to subsequent requests for the data in accordance with the cache coherency protocol as described herein.

**[0072]** FIG. 9 depicts a method that includes employing a first cache state to identify a first node of a system as being an ordering point for a block of data, as indicated

at 300. The method also includes providing a data response from the first node to a request from a second node of the system for the block of data, as indicated at 310.

**[0073]** What have been described above are examples of the present invention. It is, of course, not possible to describe every conceivable combination of components or methodologies for purposes of describing the present invention, but one of ordinary skill in the art will recognize that many further combinations and permutations of the present invention are possible. Accordingly, the present invention is intended to embrace all such alterations, modifications and variations that fall within the spirit and scope of the appended claims.